Introduction to Big Data OPPE

Name: Faizan Mulla
Roll No: 21F1003885

Problem Statement

The fraud control unit at a major regulator would like to probe historical stock trades for abnormal stock trading behaviours. To start with, any of the below events that happen should be considered an anomaly:

- A1: Current trade for a particular stock is at a price that deviates from previous minute's trade close price by more than 0.5% (over or under).
- A2: Traded volume in a particular stock is more than 2% above the average traded volume for the last 10 minutes prior.

Whenever an anomaly is found, then the trade that triggered the anomaly should be captured for further scrutiny.

Your task is to identify all anomalies, emit the trades that are anomalous along with the type of the anomaly that got detected. Note that if a trade is anomalous due to any one definition, it suffices to emit that trade without needing to check that trade for other anomalies.

Solution

The solution involves the following key steps:

1. Setting up the environment locally and on GCP.
2. Implementing the Kafka Producer to read from GCS and send data to Kafka.
3. Writing the Spark Streaming Consumer to process Kafka messages.
4. Running and verifying both components simultaneously.

Implementation Details

Step 1: Environment Setup

Local Setup

1. Install required packages:

```bash
sudo apt update; sudo apt -y upgrade
sudo apt install -y python3.11-venv openjdk-11-jdk
```

```bash
python3 -m venv .venv
source .venv/bin/activate
pip install -U kafka google-cloud-storage pyspark pandas
```

2. Download and extract Kafka:

```bash
wget https://downloads.apache.org/kafka/3.8.0/kafka_2.13-3.8.0.tgz
tar -xzf kafka_2.13-3.8.0.tgz
rm kafka_2.13-3.8.0.tgz

cd kafka_2.13-3.8.0/
```

GCP Setup

1. Create a GCS Bucket
   - Navigate to the Google Cloud Console.
   - Create a new bucket named `**ibd-oppe-bucket-2**`.
   - Upload the cleaned stock data CSV file to this bucket.

2. Dataproc Cluster Creation

   Created a Dataproc cluster with following specifications:
   - chose the option: Create cluster on compute engine
   - manager node: series → e2 // machine type → e2-standard-2 (2vCPU, 1 core, 8GB memory)
   - reduce primary disk size from 500GB to something less like 50GB.
   - exact same settings for worker nodes too.
   - Region: **asia-south-1**
   - in the customize cluster menu, **uncheck** the INTERNAL IP ONLY option.

3. Virtual Machine Creation

   A. VM instance configuration

   - Navigate to Compute Engine and then to VM instances.
   - Main settings:
     - Name: **ibd-oppe-vm**
     - Region: **asia-south-1**
     - Zone: **asia-south-1-a**

- Keep Machine Configuration settings as default (or you can change it based on your requirement)

- Identity and Access Scopes settings (**IMP**) →
  Choose this: Allow full access to all Cloud APIs

- Firewall Settings →
  Check these boxes: Allow HTTP traffic & Allow HTTPS traffic

- Click on “Create”

B. Environment Preparation / VM setup

Click the "**SSH**" button next to your VM. This opens a browser-based terminal.

Create Virtual Environment

- First install required packages:
  sudo apt-get update
  sudo apt -y upgrade
  sudo apt install -y python3.12-venv

- Create a virtual environment
  python3 -m venv .venv

- Activate the virtual environment:
  source .venv/bin/activate

C. Installing Java and Setting JAVA_HOME

Since PySpark requires Java, you need to install Java and set the JAVA_HOME environment variable.

- Install OpenJDK 11:

  sudo apt update
  sudo apt install -y openjdk-11-jdk

- Verify Java Installation:

  java -version

- Find Java Installation Path: *readlink -f $(which java)*
- Remove */bin/java* to get JAVA_HOME

  Example: */usr/lib/jvm/java-11-openjdk-amd64*

- Set JAVA_HOME Environment Variable:

  - Open the shell configuration file in a text editor:
    nano ~/.bashrc

  - Add the following lines at the end of the file:
    export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
    export PATH=$JAVA_HOME/bin:$PATH

- Save and exit the editor (Ctrl + O, then Ctrl + X).
- Apply the Changes: *source ~/.bashrc*
- Verify JAVA_HOME: *echo $JAVA_HOME*

Step 2: Kafka Producer Implementation

1. Producer Script (`kafka_producer.py`):

```python
from kafka import KafkaProducer
import pandas as pd
import time
import json


file_path = "gs://ibd-oppe-bucket-2/cleaned_stock_data.csv"
topic="stock_data"

batch_size = 10

# Kafka producer setup
producer = KafkaProducer(
    bootstrap_servers="localhost:9092",
    value_serializer=lambda v: json.dumps(v).encode("utf-8"),
)


try:
    df = pd.read_csv(file_path)

    records_processed = 0

    total_records = len(df)
    print(f"Preparing to send {total_records} records to Kafka...")

    while records_processed < total_records:
        batch_end = min(records_processed + batch_size, total_records)
        batch = df.iloc[records_processed:batch_end].to_dict("records")

        print(
            f"\nSending batch of {len(batch)} records (records {records_processed + 1} to {batch_end})..."
        )
        producer.send(topic, batch)

        records_processed += len(batch)
        print(f"Total records processed so far: {records_processed}/{total_records}")

        if records_processed < total_records:
            print("Sleeping for 10 seconds")
            time.sleep(10)

    print("\nAll records processed and sent to Kafka successfully!")

except Exception as e:
    print("An error occurred:", e)
```

Step 3: Spark Consumer Implementation

1. Consumer Script (`spark_consumer.py`):

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import avg, col, lag
from pyspark.sql.window import Window
from pyspark.sql.types import StructType, StructField, StringType, DoubleType


spark = SparkSession.builder.appName("stock_anomaly_detection").getOrCreate()

df = (
    spark.readStream.format("kafka")
    .option("kafka.bootstrap.servers", "localhost:9092")
    .option("subscribe", "stock_data")
    .load()
)

schema = StructType(
    [
        StructField("timestamp", StringType(), True),
        StructField("company", StringType(), True),
        StructField("close", DoubleType(), True),
        StructField("volume", DoubleType(), True),
    ]
)

stock_data = df.selectExpr("CAST(value AS STRING) as json_string")

window_spec = Window.partitionBy("company").orderBy("timestamp").rowsBetween(-10, -1)

stock_data = (
    stock_data
        .withColumn(
            "rolling_avg_volume",
            avg(col("volume")).over(window_spec)
        )
        .withColumn(
            "prev_close",
            lag(col("close")).over(Window.partitionBy("company").orderBy("timestamp")),
        )
        .withColumn(
            "price_change_percent", (col("close") - col("prev_close")) / col("prev_close") * 100
        )
        .withColumn(
            "anomaly_flag",
            (col("price_change_percent") > 0.5) | (col("volume") > col("rolling_avg_volume") * 1.02),
        )
)

anomalies = stock_data.filter(col("anomaly_flag"))
query = anomalies.writeStream.outputMode("append").format("console").start()

query.awaitTermination()
```

Step 4: Execution

In order to open a terminal, go to the running **VM**. Now, click on **SSH**, on the top left corner.

1. **Start Kafka (Zookeeper server + Kafka server)**

   Open **Terminal 1**:

   ```bash
   source .venv/bin/activate ; cd kafka_2.13-3.8.0/
   bin/zookeeper-server-start.sh config/zookeeper.properties
   ```

   Open **Terminal 2**:

   ```bash
   source .venv/bin/activate ; cd kafka_2.13-3.8.0/
   bin/kafka-server-start.sh config/server.properties
   ```

   Create Kafka Topic: (*name*=stock_data)

   ```bash
   bin/kafka-topics.sh --create --topic sales_data --bootstrap-server localhost:9092
   --partitions 1 --replication-factor 1
   ```

2. Start **producer service** for `stock_data` topic

   Open **Terminal 3:**

   ```bash
   source .venv/bin/activate ; cd kafka_2.13-3.8.0/
   bin/kafka-console-producer.sh --topic stock_data --broker-list localhost:9092
```

```

3. Run **Kafka Producer**

    Open **Terminal 6:**

    - Upload the ***kafka_producer.py*** file using “Upload Files”
    - Now run this file using:

    ```bash
    *source .venv/bin/activate*
    *python3 kafka_producer.py*
    ```

4. Start **consumer service** for `stock_data` topic

    Open **Terminal 4:**

    ```bash
    *source .venv/bin/activate ; cd kafka_2.13-3.8.0/*

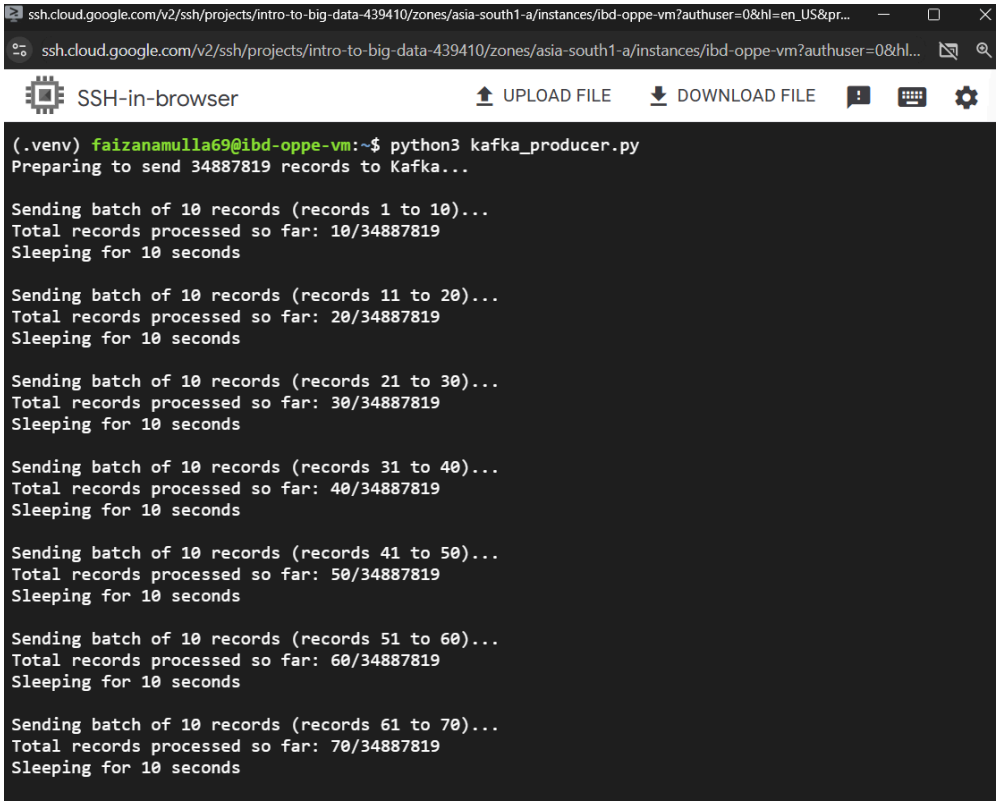
    *bin/kafka-console-consumer.sh --topic stock_data --bootstrap-server localhost:9092*
    ```
    ```

Results

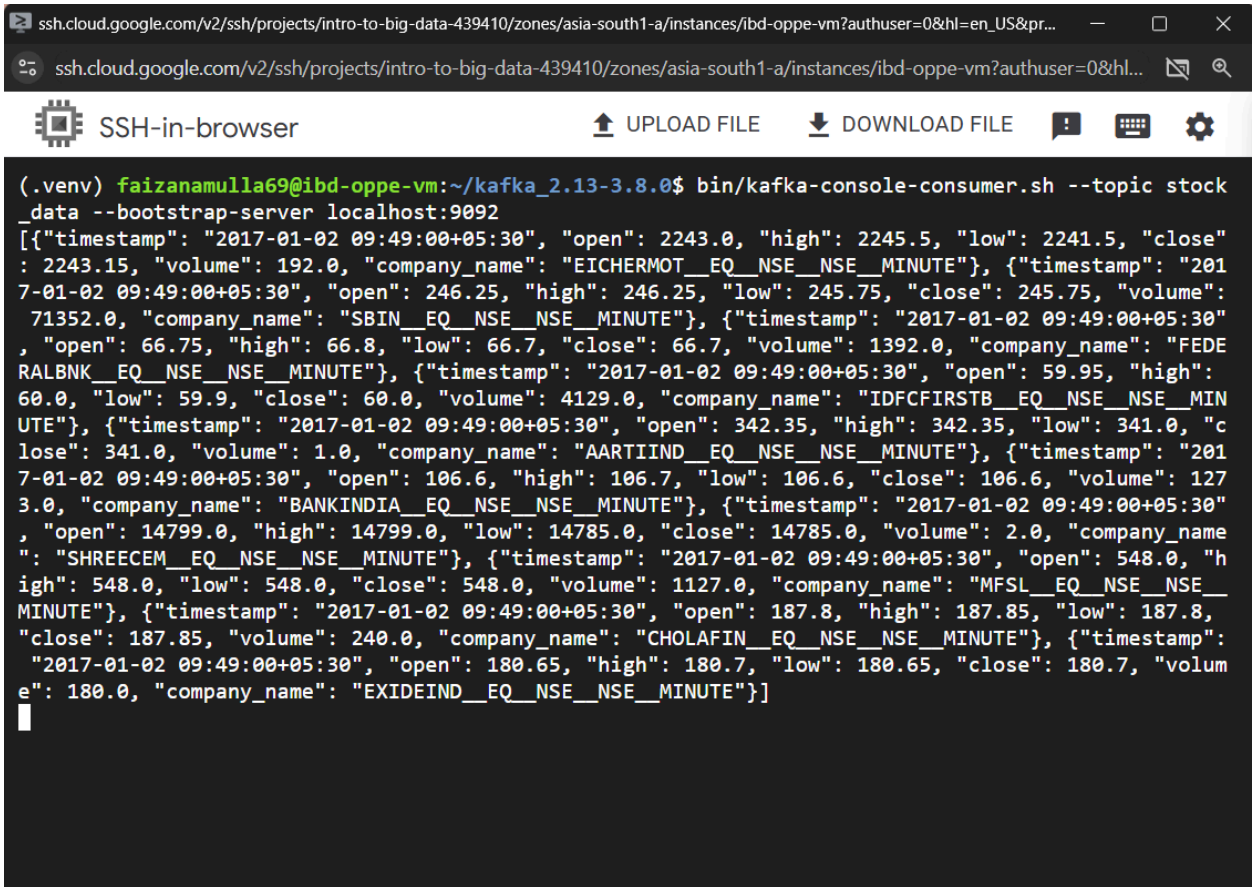
1. The Kafka Producer sent all rows from the cleaned_stock_data.csv file in batches of 10, with a 10-second gap between batches.
2. The Spark Consumer read the data from the Kafka topic stock_data and processed it in real-time.
3. Anomalies were identified based on price changes and volume spikes, and the results were shown on the Dataproc console.
4. The process stopped automatically after all the data was sent and processed.

Relevant Screenshots

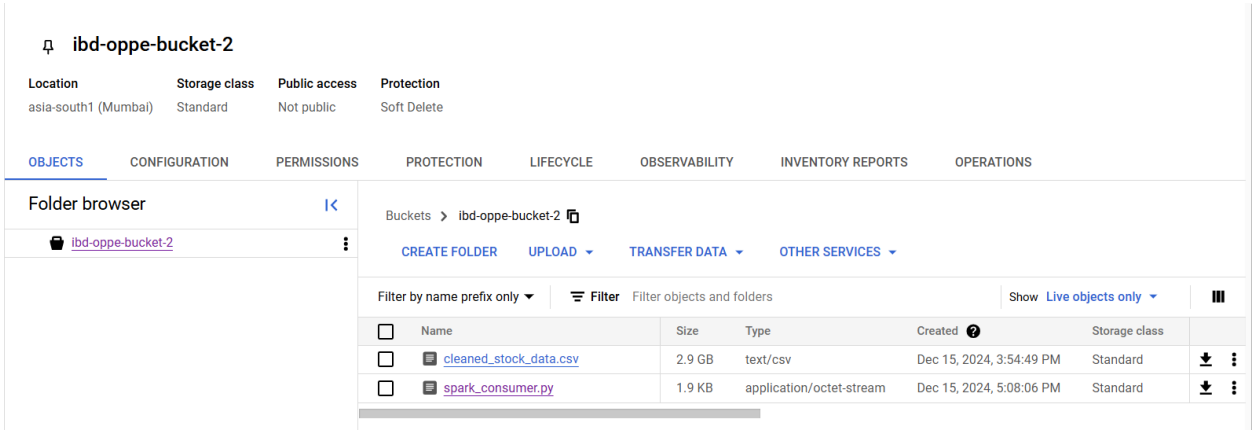
1. Kafka Producer Output



2. Spark Consumer Output



3. GCS Bucket:



4. Dataproc Cluster: